Urgent Partial Order Reduction for Extended Timed Automata

Kim G. Larsen, Marius Mikučionis, Marco Muñiz, and Jiří Srba

Department of Computer Science, Aalborg University, Denmark
{kgl,marius,muniz,srba}@cs.aau.dk

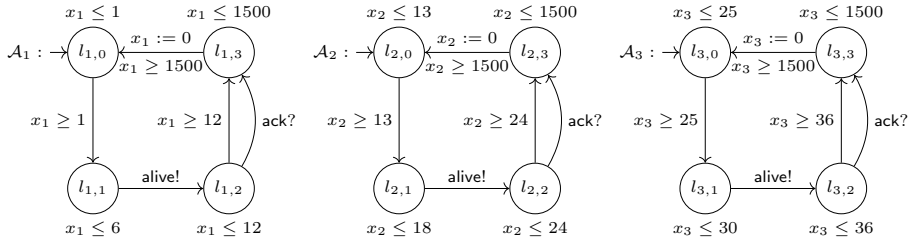
Abstract. We propose a partial order reduction method for reachability analysis of networks of timed automata interacting via synchronous channel communication and via shared variables. Our method is based on (classical) symbolic delay transition systems and exploits the urgent behavior of a system, where time does not introduce dependencies among actions. In the presence of urgent behavior in the network, we apply partial order reduction techniques for discrete systems based on stubborn sets. We first describe the framework in the general setting of symbolic delay time transition systems and then instantiate it to the case of timed automata. We implement our approach in the model checker Uppaal and observe a substantial reduction in the reachable state space for case studies that exhibit frequent urgent behaviour and only a moderate slowdown on models with limited occurence of urgency.

1 Introduction

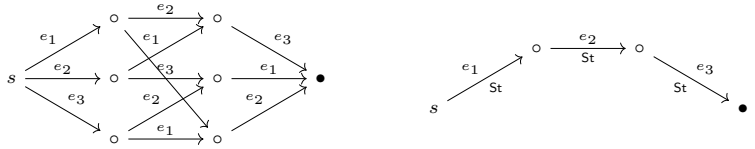
Partial order reduction techniques [4] based on persistent sets [14], ample sets [22] or stubborn sets [17,27] have proved beneficial for the state space exploration of systems that exhibit high degree of concurrency. As many actions in such systems can be (in a syntax-driven manner) considered as independent, these techniques will explore only a subset of the possible interleavings of independent actions while preserving the property of the system we are interested in.

The techniques of partial order reductions for untimed system have only recently been extended to *timed* systems with indication of success. For more than two decades timed systems have resisted several partial order reduction attempts, largely caused by the fact that time introduces additional dependencies between actions that will normally be considered as independent. In [15] the authors show a potential for stubborn reductions for networks of timed automata, however using only approximate abstraction approach. A new idea of exploiting urgency in timed systems in order to facilitate efficient partial order reduction appeared in [8] in the context of timed-arc Petri nets with discrete time.

We take the idea of urgency-based [23] partial order reduction one step further and extend the method towards the case of networks of extended timed automata in the Uppaal style, including handshake and broadcast communication primitives, communication over shared variables as well as a C-like imperative programming language allowing for complex computation over discrete



(a) Three sensors of the simplified fire alarm system



(b) Fragment of the full and reduced transition systems for the system in Figure 1a

Fig. 1: Simplified Fire Alarm System

structured variables. Our main contribution is a partial order reduction method for urgent behavior based on the classical (zone-based) symbolic semantics for networks of timed automata and its efficient implementation in the industrial strength real-time verification tool Uppaal. An additional challenge is to develop static analysis for the rich modeling language of Uppaal and combine it with symbolic model checking techniques in a sound way. On a number of experiments we show the applicability of the proposed method w.r.t. state-space and time reduction.

Fire Alarm System Example. To illustrate the effect of our urgency-based partial order reduction technique, we consider a simplified version of an industrial fire alarm system [11]. The system uses a communication protocol based on the Time Division Multiple Access (TDMA) paradigm, and has over 100 sensors each of them assigned a unique time slot for sending and receiving messages. Figure 1a shows a down-scaled and simplified version of the system with three sensors, each modeled as a timed automaton. Each sensor has its own clock x_i, with the corresponding TDMA slot modeled by guards in ($x_i \ge 1500$) and invariants ($x_i \le 1500$). At the end of the TDMA cycle i.e. when $x_i = 1500$ every sensor resets its clock and goes back to its initial location. Figure 1b left shows the fragment of the reachable transition system starting at the configuration $s = ((l_{1,3}, l_{2,3}, l_{3,3}), x_1 = x_2 = x_3 = 1500)$ where time progress is disabled due to the invariants $x_i \le 1500$. The transitions are induced by the edges $e_i = \langle l_{i,3}, \tau, x_i \ge 1500, x_i = 0, l_{i,0} \rangle$. States of the form $\circ$ denote the so-called zero time states where time cannot progress, whereas the filled state $\bullet$ denotes a situation where time can delay. Figure 1b right shows the corresponding reduced transition

system that contains only one interleaving sequence that allows us to reach the state where time can delay again.

Related Work. The most related work in [8] presents an urgent partial order reduction method for *discrete* time systems based on stubborn set construction [17,27]. The method is instantiated to timed-arc Petri nets and compared to our case, it does not consider discrete data structures nor any communication primitives. In our work we focus on *continuous* time systems modeled as networks of timed automata, requiring us to use symbolic transition system as the underlying semantic model. The idea of applying partial order reduction for independent events that happen at the same time also appeared in [9] however this methods is not as efficient as ours because it is static (precomputed before the state space exploration). In our approach we apply a dynamic reduction that on-the-fly identifies independent actions even in the presence of communication between the components, possibly sharing some resources.

Partial order reduction techniques applied to timed automata [2] include the early works [7,19,10] based on the notion of local and global clocks or the concept of covering as generalized dependencies. However, there is not provided any experimental evaluation of the proposed techniques. There exist also techniques based on event zones [18,21] and on merging zones from different interleaved executions [25]. These are exact techniques comparable to approximate convex-hull abstraction which is by now superseded by the exact LU-abstraction [5]. More recently, over-approximative methods based on abstracted zone graphs were also studied in [15]. The main difference is that our approach is an exact method that is applicable directly to the state-of-the-art techniques implemented in Uppaal.

Finally, *quasi-equal clocks* [16] are clocks for which in all computations their values are equal or if one clock gets reset then a reset must urgently eventually occur also for the other clocks, assuming that resets occur periodically. Reductions using quasi-equal clocks yield exponential savings and have been used to verify a number of industrial systems. However, this approach is based on syntactic transformations and requires a method for detecting quasi-equal clocks [20]. Our approach fully automatizes reductions based on quasi-equal clocks and further generalizes to scenarios where clock resets have irregular reset periods.

2 Partial Order Reduction for Symbolic Delays

We describe the general idea of our partial order reduction technique in terms of symbolic delay transition systems. Intuitively a symbolic delay corresponds to time elapsing in the zone graph for timed automata or flow in the region graph of a hybrid system. Let A be a set of actions and δ a symbolic delay with $A \cap \{\delta\} = \emptyset$.

Definition 1 (Symbolic Delay Transition System). *A* symbolic delay transition system *is a tuple* $(S, s_0, \rightarrow)$ *where* S *is a set of states,* $s_0 \in S$ *is the initial state, and* $\rightarrow \subseteq S \times (A \cup \{\delta\}) \times S$ *is the transition relation.*

If $(s, \alpha, s') \in \rightarrow$ we write $s \xrightarrow{\alpha} s'$. In this paper we consider only deterministic systems: a transition system is *deterministic* if $s \xrightarrow{\alpha} s'$ and $s \xrightarrow{\alpha} s''$ implies $s' = s''$. For the rest of this section, let us assume a fixed symbolic delay transition system $(S, s_0, \rightarrow)$ and a set of goal states $G \subseteq S$.

A state $s \in S$ is *zero time* if it can not delay, denoted by $\mathsf{zt}(s)$ and defined by $\mathsf{zt}(s)$ iff $\forall s' \in S, \alpha \in A \cup \{\delta\}.\ s \xrightarrow{\alpha} s' \implies \alpha \in A$. A *reduction* is a function $\mathsf{St} : S \to 2^A$. A reduced transition relation is a relation $\xrightarrow[\mathsf{St}]{} \subseteq \rightarrow$ such that $s \xrightarrow[\mathsf{St}]{\alpha} s'$ iff $s \xrightarrow{\alpha} s'$ and $\alpha \in \mathsf{St}(s) \cup \{\delta\}$. For a given state $s \in S$ we define $\overline{\mathsf{St}(s)} \stackrel{\mathsf{def}}{=} A \setminus \mathsf{St}(s)$ to be the set of all actions not in $\mathsf{St}(s)$. Given a sequence of labels $w = \alpha_1\alpha_2\alpha_3 \ldots \alpha_n \in (A \cup \{\delta\})^*$ we write $s \xrightarrow{w} s'$ iff $s \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s'$. If a sequence w of length n is such that $s \xrightarrow{w} s'$ we also write $s \rightarrow^n s'$. The set of *enabled actions* at state $s \in S$ is $\mathsf{En}(s) \stackrel{\mathsf{def}}{=} \{a \in A \mid \exists s' \in S.\ s \xrightarrow{a} s'\}$.

The *reachability problem*, given a symbolic delay transition system $(S, s_0, \rightarrow)$ and a set of goal states G, is to decide whether there is $s' \in G$ such that $s_0 \rightarrow^* s'$.

Definition 2 (Reachability Preserving Reduction). *A reduction* St *is* reachability preserving *if it satisfies the following conditions:*

$(\mathcal{Z})$ $\forall s \in S.\ \neg\mathsf{zt}(s) \implies \mathsf{En}(s) \subseteq \mathsf{St}(s)$
$(\mathcal{D})$ $\forall s, s' \in S.\ \forall w \in \overline{\mathsf{St}(s)}^*.\ \mathsf{zt}(s) \wedge s \xrightarrow{w} s' \implies \mathsf{zt}(s')$
$(\mathcal{R})$ $\forall s, s' \in S.\ \forall w \in \overline{\mathsf{St}(s)}^*.\ \mathsf{zt}(s) \wedge s \xrightarrow{w} s' \wedge s \notin G \implies s' \notin G$
$(\mathcal{W})$ $\forall s, s' \in S.\ \forall w \in \overline{\mathsf{St}(s)}^*.\ \forall a \in \mathsf{St}(s).\ \mathsf{zt}(s) \wedge s \xrightarrow{wa} s' \implies s \xrightarrow{aw} s'$

If a delay is possible at state s Condition $\mathcal{Z}$ will ensure that there is no reduction. Condition $\mathcal{D}$ ensures that states which can delay are preserved. Condition $\mathcal{R}$ ensures that goal states are preserved and finally Condition $\mathcal{W}$ corresponds to the classical stubborn set requirement that stubborn actions can be commuted to the beginning of the execution. The following theorem was proved in [8] for the case of timed transitions systems.

Theorem 1 (Reachability Preservation). *Let* St *be a reachability preserving reduction. Let* $s \in S$ *and* $s \rightarrow^n s'$ *for some* $s' \in G$ *then* $s \xrightarrow[\mathsf{St}]{}^m s''$ *for some* $s'' \in G$ *where* $m \leq n$.

3 Extended Timed Automata (XTA)

We apply our method to the theory of timed automata [2]. Our formal model is *extended timed automata* and it is an abstract representation of modeling formalism used in the tool UPPAAL [6].

Clocks and Discrete Variables. Let X be a set of *clocks*. A *clock valuation* is a function $\mu : X \to \mathbb{R}_{\geq 0}$. We use $\mathcal{V}(X)$ to denote the sets of all valuations for clocks in X. Let V be a set of *discrete variables*. The function D assigns to each variable $v \in V$ a finite domain $D(v)$. A *variable valuation* is a function $\nu : V \to \bigcup_{v \in V} D(v)$ that maps variables to values such that $\nu(v) \in D(v)$. We use $\mathcal{V}(V)$ to denote the set of all variable valuations. We let μ_0 resp. ν_0 to denote the valuation that maps every clock resp. variable to the value 0.

Expressions. We use expr to denote an expression over V. We assume that expressions are well typed and for expression expr we use $D(\mathsf{expr})$ to denote its domain. Given a variable valuation ν and an expression expr, we use $\mathsf{expr}^\nu \in D(\mathsf{expr})$ to denote the value of expr under ν. We use $V(\mathsf{expr}) \in 2^V$ to denote the set of variables in expr such that for all $v \in V(\mathsf{expr})$ and for all $\nu, \nu' \in \mathcal{V}(V)$ if $\nu(v) = \nu'(v)$ then $\mathsf{expr}^\nu = \mathsf{expr}^{\nu'}$.

Constraints. The set $B(X)$ is the set of *clock constraints* generated by the grammar $\phi ::= x \bowtie \mathsf{expr} \mid \phi_1 \wedge \phi_2$, where $x \in X$, $D(\mathsf{expr})$ is the domain of all natural numbers $\mathbb{N}$ and $\bowtie \in \{<, \leq, \geq, >\}$. The set $B(V)$ is a set of *Boolean variable constraints* over V. The set $B(X, V)$ of constraints comprises $B(X)$, $B(V)$, and conjunctions over clock and variable constraints. Given a constraint $\phi \in B(X, V)$, we use $X(\phi)$ to denote the set of clocks in ϕ, and $V(\phi)$ to denote the set of variables in ϕ. We define the evaluation of a constraint $\phi \in B(X, V)$ as ϕ^ν where expressions in ϕ are evaluated under ν.

Updates. A *clock update* is of the form $x := \mathsf{expr}$ where $x \in X$, and $D(\mathsf{expr}) = \mathbb{N}$. A *variable update* is of the form $v := \mathsf{expr}$ where $v \in V$ and $D(v) = D(\mathsf{expr})$. The set $U(X, V)$ of *updates* contains all finite, possibly empty sequences of clock and variable updates. Given clock valuation $\mu \in \mathcal{V}(X)$, variable valuation $\nu \in \mathcal{V}(V)$, and update $r \in U(X, V)$, we use r^ν to denote the update resulting after evaluating all expressions in r under ν, we use $X(r)$ to denote the set of clocks in r, and $V(r)$ to denote the set of variables in r. We let $[\![r^\nu]\!] : \mathcal{V}(X) \cup \mathcal{V}(V) \to \mathcal{V}(X) \cup \mathcal{V}(V)$ be a map from valuations to valuations. We use $\mu[r^\nu]$ to denote the updated clock valuation $[\![r^\nu]\!](\mu)$. Analogously, for variable valuation ν', we use $\nu'[r^\nu]$ to denote the updated variable valuation $[\![r^\nu]\!](\nu')$.

Channels. Given a set C of *channels*, the set $H(C)$ of synchronizations over channels is generated by the grammar $h ::= c[\mathsf{expr}]! \mid c[\mathsf{expr}]? \mid \tau$, where $c \in C$, $D(\mathsf{expr}) = \mathbb{N}$, and τ represents an internal action. Given a variable valuation ν, for synchronization h of the form $c[\mathsf{expr}]!$ we use h^ν to denote $c[\mathsf{expr}^\nu]!$, and similar for synchronizations of the form $c[\mathsf{expr}]?$.

Definition 3 (Extended Timed Automata XTA). *A* extended timed automaton $\mathcal{A}$ *is a tuple* $(L, L^u, L^c, l_0, X, V, H(C), E, I)$ *where:* L *is a set of locations,* $L^u \subseteq L$ *denotes the set of urgent locations in* L, $L^c \subseteq L$ *denotes the set of committed locations in* L *and* $L_u \cap L_c = \emptyset$, $l_0 \in L$ *is the initial location,* X *is a nonempty the set of clocks,* V *is the set of variables,* $H(C)$ *is a set of channels expressions for set of channels* C, $E \subseteq L \times H(C) \times B(X) \times B(V) \times U(X, V) \times L$ *is a set of edges between locations with a channel expressions, a clock guard, a variable guard, an update set, and* $I : L \to B(X)$ *assigns clock invariants to locations.*

Definition 4 (Network of XTA). *A network* $\mathcal{N}$ *of XTA consists of a finite sequence* $\mathcal{A}_1, \ldots, \mathcal{A}_n$ *of XTA, where* $\mathcal{A}_i = (L_i, L_i^u, L_i^c, l_i^0, X_i, V_i, H(C)_i, E_i, I_i)$ *for* $1 \leq i \leq n$. *Locations are pairwise disjoint i.e.* $L_i \cap L_j = \emptyset$ *for* $1 \leq i, j \leq n$

and $i \neq j$. The set of locations is $L = \cup_{i=1}^{n} L_i$, analogously for urgent L^u and committed L^c locations. The set of clocks is $X = \cup_{i=1}^{n} X_i$ and the set of variables is $V = \cup_{i=1}^{n} V_i$. The set of channel expressions is $H(C) = \cup_{i=1}^{n} H(C)_i$. The set of edges is $E = \cup_{i=1}^{n} E_i$. A location vector is a vector $\boldsymbol{l} = (l_1, \ldots, l_n)$, and $\boldsymbol{l}_0 = (l_1^0, \ldots, l_n^0)$ is the initial location vector. The invariant function over location vectors is $I(\boldsymbol{l}) = \bigwedge_i I_i(l_i)$.

We write $\boldsymbol{l}[l_i'/l_i]$ to denote the vector where the i-th element l_i of $\boldsymbol{l}$ is replaced by l_i'. We write $\boldsymbol{l}^i$ to denote the i-th element of $\boldsymbol{l}$.

Zones. We assume the canonical satisfaction relation "$\models$" between valuations and constraints in $B(X)$ and $B(V)$. The set $B^+(X)$ of *extended clock constraints* is generated by the grammar $\phi ::= x \bowtie c \mid \phi_1 \wedge \phi_2 \mid x - y \bowtie c$, where $x, y \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, \geq, >\}$. A *zone* $[\![Z]\!]$ is a set of clock valuations described by an extended clock constraint $Z \in B^+(X)$ where $[\![Z]\!] \stackrel{\text{def}}{=} \{\mu \in \mathcal{V}(X) \mid \mu \models Z\}$. When it is clear from the context, we use Z and $[\![Z]\!]$ interchangeably. We define $Z^{\uparrow} \stackrel{\text{def}}{=} \{\mu + d \mid \mu \in Z, d \in \mathbb{R}_{\geq 0}\}$, where for $d \in \mathbb{R}_{\geq 0}$, $\mu + d$ maps each clock $x \in X$ to the value $\mu(x) + d$. For zone Z and update r we define $Z[r] \stackrel{\text{def}}{=} \{\mu[r] \mid \mu \in Z\}$.

For timed automata we consider the set of actions $A = 2^E$ that corresponds to the discrete transitions induced by the edges E, and δ is the delay action induced by non-zero delay transitions. We can now define the symbolic semantics of networks of timed automata in terms of a zone graph (see e.g. [1]).

Definition 5 (Semantics of a Network of XTA). *Let $\mathcal{N} = \mathcal{A}_1, \ldots, \mathcal{A}_n$ be a network of TA. Its semantics is defined as a* symbolic delay transition system *(zone graph) $(S, s_0, \rightarrow)$, where $S \subseteq (L_i \times \cdots \times L_n) \times B^+(X) \times \mathcal{V}(V)$ is the set of states comprising a location vector, a zone, and a variable valuation, $s_0 = (\boldsymbol{l}_0, \{\mu_0\}, \nu_0)$ is the initial state, and $\rightarrow \subseteq S \times (A \cup \{\delta\}) \times S$ is the transition relation defined by:*

- *delay transition, $(\boldsymbol{l}, Z, \nu) \xrightarrow{\delta} (\boldsymbol{l}, Z^{\uparrow} \wedge I(\boldsymbol{l})^{\nu}, \nu)$ if $\boldsymbol{l}^i \notin L_i^u \cup L_i^c$ for $1 \leq i \leq n$, and $\exists \mu \in Z, d \in \mathbb{R}_{\geq 0}. d > 0 \wedge \mu + d \models I(\boldsymbol{l})^{\nu}$,*
- *internal transition, $(\boldsymbol{l}, Z, \nu) \xrightarrow{\{e_i\}} (\boldsymbol{l}[l_i'/l_i], Z', \nu')$ if $e_i = (l_i, \tau, \phi, \psi, r, l_i') \in E_i$ s.t. $Z' = (Z \wedge I(\boldsymbol{l})^{\nu} \wedge \phi^{\nu})[r^{\nu}] \wedge I(\boldsymbol{l}[l_i'/l_i])^{\nu'}$, where $Z' \neq \emptyset$, $\nu' = \nu[r^{\nu}]$, $\nu \models \psi^{\nu}$, and if $\boldsymbol{l}^k \in L_k^c$ for some $1 \leq k \leq n$ then $l_i \in L_i^c$,*
- *handshake transition, $(\boldsymbol{l}, Z, \nu) \xrightarrow{\{e_i, e_j\}} (\boldsymbol{l}[l_j'/l_j, l_i'/l_i], Z', \nu')$ if there exists $e_i = (l_i, h_i!, \phi_i, \psi_i, r_i, l_i') \in E_i$ and $e_j = (l_j, h_j?, \phi_j, \psi_j, r_j, l_j') \in E_j$ s.t. $h_i^{\nu} = h_j^{\nu}$, and $Z' = (Z \wedge I(\boldsymbol{l})^{\nu} \wedge \phi_i^{\nu} \wedge \phi_j^{\nu})[r_i^{\nu}][r_j^{\nu}] \wedge I(\boldsymbol{l}[l_j'/l_j, l_i'/l_i])^{\nu'}$, where $Z' \neq \emptyset$, $\nu \models (\psi_i^{\nu} \wedge \psi_j^{\nu})$, $\nu' = \nu[r_i^{\nu}][r_j^{\nu}]$, and if $\boldsymbol{l}^k \in L_k^c$ for some $1 \leq k \leq n$ then $l_i \in L_i^c$ or $l_j \in L_j^c$.*

In the following, we are given a network of TA $\mathcal{N} = \mathcal{A}_1, \ldots, \mathcal{A}_n$ with locations L, clocks X, variables V, and induced symbolic transition system $(S, s_0, \rightarrow)$.

Definition 6 (Properties). *A* formula *is given by the grammar $\phi ::= deadlock \mid l \mid x \bowtie c \mid \psi_v \mid \phi_1 \wedge \phi_2$, where $l \in L$, $x \in X$, $\bowtie \in \{<, \leq, \geq, >\}$, $c \in \mathbb{N}$, and ψ_v is*

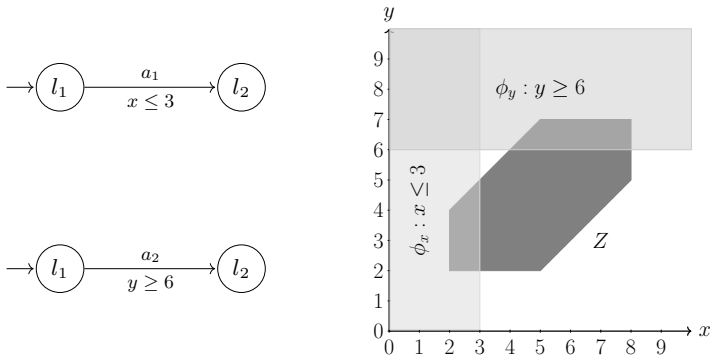


Fig. 2: Two components with actions a_1 and a_2. Actions are enabled at zone Z. Note that executing a_1 will disable a_2 and vice versa.

a Boolean constraint for $v \in V$. Let $(\boldsymbol{l}, Z, \nu) \in S$ be a state. The satisfaction of a formula is inductively defined as follows:

$$
\begin{array}{ll}
(\boldsymbol{l}, Z, \nu) \models \textit{deadlock} & \textit{iff } \exists \mu \in Z, \forall d \in \mathbb{R}_{\geq 0}. \mathsf{En}((\boldsymbol{l}, \{\mu + d\}, \nu)) = \emptyset \\
(\boldsymbol{l}, Z, \nu) \models l & \textit{iff } \boldsymbol{l}^i = l \textit{ for some } i \textit{ with } 1 \leq i \leq n \\
(\boldsymbol{l}, Z, \nu) \models x \bowtie c & \textit{iff } \exists \mu \in Z.\ \mu \models x \bowtie c \\
(\boldsymbol{l}, Z, \nu) \models \psi_v & \textit{iff } \nu \models \psi_v \\
(\boldsymbol{l}, Z, \nu) \models \phi \wedge \psi & \textit{iff } (\boldsymbol{l}, Z, \nu) \models \phi \textit{ and } (\boldsymbol{l}, Z, \nu) \models \psi
\end{array}
$$

A network satisfies ϕ iff its initial state can reach a state that satisfies ϕ.

4 Reachability Preserving Reduction for XTA

In this section we provide syntactic based sound approximations for all the elements required by our technique. In Subsection 4.1 we give a semantic definition for independence of actions, then we describe a syntactic independence relation. In Subsection 4.2 we identify the relevant actions which need to be included in the stubborn set to preserve states which can delay. Finally, in Subsection 4.3 we describe the stubborn sets for preserving goal states. For the rest of this section we are given a network $\mathcal{N} = \mathcal{A}_1, \ldots, \mathcal{A}_n$, with edges E and components $\mathcal{A}_i = (L_i, L_i^u, L_i^c, l_{i_0}, X_i, V_i, H(C)_i, E_i, I_i)$, the corresponding transition system $(S, s_0, \rightarrow)$ with actions $A = 2^E$, and state $s = (\boldsymbol{l}, Z, \nu)$.

4.1 Independence for Actions

The notion of *independence of actions* plays a key role in partial order reduction. Intuitively two actions are independent if they can not disable each other and they commute.

Definition 7 (Independence of Actions). *An* independence *relation for state $s \in S$ is a symmetric, anti-reflexive relation $\leftrightsquigarrow_s \subseteq A \times A$ satisfying the following conditions for each $(a_1, a_2) \in \leftrightsquigarrow_s$:*

1. $\forall s' \in S.\ s \xrightarrow{a_1} s' \wedge a_2 \in \mathsf{En}(s) \implies a_2 \in \mathsf{En}(s')$
2. $a_1 \in \mathsf{En}(s) \wedge a_2 \in \mathsf{En}(s) \implies \exists s' \in S.\ s \xrightarrow{a_1 a_2} s' \wedge s \xrightarrow{a_2 a_1} s'$

If $(a_1, a_2) \in \leftrightsquigarrow_s$ they are independent *at s denoted by $a_1 \leftrightsquigarrow_s a_2$. Otherwise they are* dependent *at s denoted by $a_1 \nleftrightsquigarrow_s a_2$.*

In what follows we will provide a syntactic independence relation on actions. Toward this goal, first we define operations on actions and we define a syntactic independence relation on operations.

Additional Notation. For a given edge $e = (l, h, \phi, \psi, r, l') \in E$ we use $src(e)$, $dst(e)$ to denote the source location l and the destination location l' of edge e. Given actions $a, a' \in A$, for action a we define its *preset* as $\mathsf{Pre}(a) \stackrel{\mathsf{def}}{=} \{src(e) \in L \mid e \in a\}$, and its *poset* as $\mathsf{Post}(a) \stackrel{\mathsf{def}}{=} \{dst(e) \in L \mid e \in a\}$. We use $\mathsf{Active}(a) \stackrel{\mathsf{def}}{=} \{\mathcal{A}_i \mid \mathcal{A}_i \text{ is in } \mathcal{N} \text{ and } \exists l \in \mathsf{Pre}(a).\ l \in L_i\}$ to denote the active components for a. We use $\mathsf{Parallel}(a, a') \stackrel{\mathsf{def}}{=} \mathsf{Active}(a) \cap \mathsf{Active}(a') = \emptyset$ to denote that actions a and a' correspond to different components. For convenience we define *Operations for Actions in TA*. The set of all operations is the set containing all constraints and resets i.e. Op is the power set of $B(X, V) \cup U(X, V)$. The set of *operations* for action $a \in A$ is given by, $\mathsf{Op}(a) \stackrel{\mathsf{def}}{=} \mathsf{Guard}(a) \cup \mathsf{Update}(a)$. Where the set of guards is $\mathsf{Guard}(a) \stackrel{\mathsf{def}}{=} \bigcup\{\phi \wedge \psi \wedge I(l) \wedge I(l') \mid (l, h, \phi, \psi, r, l') \in a\}$, and the set of updates is $\mathsf{Update}(a) \stackrel{\mathsf{def}}{=} \bigcup\{r \mid (l, h, \phi, \psi, r, l') \in a\}$. Given an operation $\mathsf{op} \in \mathsf{Op}$, the set of variables which op increments is given by $\mathsf{Inc}(\mathsf{op}) = \{v \in V(\mathsf{op}) | \exists r \in \mathsf{op}$ and r includes $v := v + 1$ with $D(v) = \mathbb{N}\}$. Analogously the set $\mathsf{Dec}(\mathsf{op})$ contains the variables which op decrements $\mathsf{Dec}(\mathsf{op}) = \{v \in V(\mathsf{op}) | \exists r \in \mathsf{op}$ and r includes $v := v - 1$ with $D(v) = \mathbb{N}\}$. The clocks and variables the operation writes is given by $\mathsf{Write}(\mathsf{op}) \stackrel{\mathsf{def}}{=} \bigcup_{r \in \mathsf{op}'}\{xv \mid r \in U(X, V)$ and $xv := \mathsf{expr}$ is in $r\}$, where op' is obtained from op by removing increment and decrement updates, formally $\mathsf{op}' = \mathsf{op} \setminus \{xv := \mathsf{expr} \in \mathsf{op} \mid \mathsf{expr}$ is of the form $xv + 1$ or $xv - 1\}$ The set $\mathsf{Readleq}(\mathsf{op}) = \{xv \in X \cup V \mid xv \leq \mathsf{expr} \in \mathsf{op}$ or $xv < \mathsf{expr} \in \mathsf{op}\}$ is the set containing clock and variables which appear in less and equal comparisons. Analogously the set $\mathsf{Readgeq}(\mathsf{op})$ contains clock and variables which appear in greater and equal comparisons in op. The clocks and variables the operation reads is given by $\mathsf{Read}(\mathsf{op}) \stackrel{\mathsf{def}}{=} X(\mathsf{op}') \cup V(\mathsf{op}')$ where op' is obtained from op by removing less (greater) and equal comparisons, formally $\mathsf{op}' = \mathsf{op} \setminus \{xv \bowtie \mathsf{expr} \in \mathsf{op} \mid \bowtie \in \{\leq, <, >, \geq\}\}$. Note that given a zone, a clock constraint can modify (write to) other clocks. Finally $\Gamma_x(Z) \stackrel{\mathsf{def}}{=} \{\mu(x) \mid \mu \in Z\}$ is the set of real values for clock x in zone Z.

Definition 8 (Independence of Operations). *Given operations $\mathsf{op}_1, \mathsf{op}_2 \in \mathsf{Op}$ and state s, operation op_1 is independent of operation op_2 at s denoted by*

$\mathsf{op}_1 \not\rightsquigarrow^\sharp_s \mathsf{op}_2$ *iff the following hold:*
(1) $\mathsf{Read}(\mathsf{op}_1) \cap (\mathsf{Write}(\mathsf{op}_2) \cup \mathsf{Inc}(\mathsf{op}_2) \cup \mathsf{Dec}(\mathsf{op}_2)) = \emptyset$
(2) $\mathsf{Readleq}(\mathsf{op}_1) \cap (\mathsf{Write}(\mathsf{op}_2) \cup \mathsf{Inc}(\mathsf{op}_2)) = \emptyset$
(3) $\mathsf{Readgeq}(\mathsf{op}_1) \cap (\mathsf{Write}(\mathsf{op}_2) \cup \mathsf{Dec}(\mathsf{op}_2)) = \emptyset$
(4) $\mathsf{Write}(\mathsf{op}_1) \cap (\mathsf{Write}(\mathsf{op}_2) \cup \mathsf{Inc}(\mathsf{op}_2) \cup \mathsf{Dec}(\mathsf{op}_2)) = \emptyset$
(5) $\mathsf{Inc}(\mathsf{op}_1) \cap (\mathsf{Write}(\mathsf{op}_2) \cup \mathsf{Dec}(\mathsf{op}_2)) = \emptyset$
(6) $\mathsf{Dec}(\mathsf{op}_1) \cap (\mathsf{Write}(\mathsf{op}_2) \cup \mathsf{Inc}(\mathsf{op}_2)) = \emptyset$
(7) $\{x \mid x \in X(\mathsf{op}_1) \cup X(\mathsf{op}_2)$ *and* $|\Gamma_x(Z)| \neq 1$ *and* $\mathsf{op}_1, \mathsf{op}_2 \in B(X,V)\} = \emptyset$
If $\mathsf{op}_1 \not\rightsquigarrow^\sharp_s \mathsf{op}_2$ *and* $\mathsf{op}_2 \not\rightsquigarrow^\sharp_s \mathsf{op}_1$ *the we write* $\mathsf{op}_1 \leftrightsquigarrow^\sharp_s \mathsf{op}_2$ *and say that* op_1 *and* op_2 *are independent at* s*. We write* $\mathsf{op}_1 \not\leftrightsquigarrow^\sharp_s \mathsf{op}_2$ *iff* op_1 *and* op_2 *are dependent.*

Intuitively two operations are independent if they read and write in different variables, note that increments and decrements are treated specially. Additionally for timed automata we need to consider that applying a guard affects a number of clocks. As an example consider Figure 2, we have that $Z \cap \phi_x \neq \emptyset$ and $Z \cap \phi_y \neq \emptyset$. However, if we apply ϕ_x we have that $(Z \cap \phi_x) \cap \phi_y = \emptyset$ this will cause the corresponding actions to disable each other. Condition (7) is not satisfied for clocks x or y in zone Z. Therefore we have $\phi_x \not\leftrightsquigarrow^\sharp_s \phi_y$. Note that Condition (7) is rather strong, since only zones which are lines or points will satisfy it (which is often the case in urgent states), relaxing this condition is subject of future work. Given two independent operations, we can conclude with a number of rules which are useful for showing that two actions do not disable each other and commute in extended timed automata.

Lemma 1. *Given state* $s = (\boldsymbol{l}, Z, \nu)$*, constraints* $\phi, \phi' \in B(X)$*, update* $r \in U(X)$*, and variable valuations* $\nu, \nu_1 \in \mathcal{V}(V)$*. The following hold:*
(1) *if* $\phi \not\leftrightsquigarrow^\sharp_s r$ *then* $[\![(Z \wedge \phi^\nu)[r^\nu]]\!] = [\![Z[r^\nu] \wedge \phi^\nu]\!]$
(2) *if* $[\![(Z \wedge \phi^\nu)[r^\nu] \wedge \phi^\nu]\!] \neq \emptyset$ *then* $[\![(Z \wedge \phi^\nu)[r^\nu] \wedge \phi^\nu]\!] = [\![(Z \wedge \phi^\nu)[r^\nu]]\!]$
(3) *if* $\phi \not\leftrightsquigarrow^\sharp_s \phi'$ *and* $\mu \in [\![Z \wedge (\phi')^\nu]\!]$ *and* $[\![Z \wedge \phi^{\nu_1}]\!] \neq \emptyset$ *then* $\mu \in [\![(Z \wedge \phi^{\nu_1})]\!]$
(4) *if* $\forall x \in X(\phi).|\Gamma_x(Z)| = 1$ *and* $[\![Z \wedge \phi^\nu]\!] \neq \emptyset$ *and* $[\![Z \wedge \phi^{\nu_1}]\!] \neq \emptyset$ *then* $[\![Z \wedge \phi^\nu]\!] = [\![Z \wedge \phi^{\nu_1}]\!]$

Lemma 1 (1) states that if a reset is independent of a constraint then the reset does not affect the constraint. Lemma 1 (2) does not require independent operations, in our proofs it is used to remove redundant application of invariants from components which are not involved in transitions. Lemma 1 (3) implicitly uses Condition (7) from Definition 8 to show that a valuation satisfying a guard is preserved after applying another guard. Lemma 1 (4) states that if a guard ϕ has been updated (via increment or decrement which in our case always produce a "bigger" constraint), then because of the shape of the zone the intersections will produce the same set.

Definition 9 (Syntactic Independence of Actions). *Given a state* $s = (\boldsymbol{l}, Z, \nu)$ *with* $\boldsymbol{l} = (l_1, \ldots, l_n)$ *and two actions* $a_1, a_2 \in A^\sharp_s$*. Actions* a_1 *and* a_2 *are* syntactically independent *at state* s *denoted by* $a_1 \not\leftrightsquigarrow^\sharp_s a_2$ *if and only if the*

following conditions hold:
(Ind1) $\mathsf{Pre}(a_1) \cap \mathsf{Pre}(a_2) = \emptyset$
(Ind2) $\exists l \in F(a_1).\ l \in L^c \iff \exists l \in F(a_2).\ l \in L^c$ *for* $F \in \{\mathsf{Pre}, \mathsf{Post}\}$.
(Ind3) $\forall op_1 \in \mathsf{Op}(a_1), op_2 \in \mathsf{Op}(a_2).\ op_1 \nleftrightsquigarrow^\sharp_s op_2$
(Ind4) $\forall i \in \{1,2\}, op \in \mathsf{Op}(a_i), j \in \{1,\ldots,n\}.\ \boldsymbol{l}^j \notin \mathsf{Pre}(a_i) \implies op \nleftrightsquigarrow^\sharp_s I(\boldsymbol{l}^j)$

Condition (Ind1) ensures that the source locations for the actions are disjoint. Condition (Ind2) takes into account the semantics of committed locations and prevents actions from disabling each other. Condition (Ind3) ensures that all the operations on the actions are independent. Finally Condition (Ind4) ensures that the operations in actions a_i for $i \in \{1,2\}$ do not modify the invariant of other components which could disable action a_{3-i}. When these syntactic conditions are satisfied we have the following theorem.

Theorem 2. *Given a zero time state $s \in S$ and two actions $a_1, a_2 \in A^\sharp_s$. If $a_1 \nleftrightsquigarrow^\sharp_s a_2$ then $a_1 \nleftrightsquigarrow_s a_2$.*

Our analysis uses the current state $s = (\boldsymbol{l}, Z, \nu)$ to conclude if two actions are independent at s. In particular we use the zone Z in Definition 8 Condition (7) to detect clock constraint dependencies. Due to this condition we can make assumptions about the shape of the zone Z which allow us to conclude that if the actions were syntactically independent at s then so they are in states reachable via independent actions.

Corollary 1. *Given state s, action $a \in A^\sharp_s$, and $A' = \{a' \in A^\sharp_s \mid a \rightsquigarrow^\sharp_s a'\}$. Then $\forall s' \in S. a' \in (A^\sharp_s \setminus A'), w \in (A^\sharp_s \setminus A')^*.\ \mathsf{zt}(s) \wedge s \xrightarrow{w} s' \xrightarrow{a'} s'' \implies a \nleftrightsquigarrow^\sharp_{s'} a'$.*

4.2 Preserving Non-Zero Time States

In order to satisfy Condition $\mathcal{D}$ from Definition 2, which ensures that the reduction preserves states that can delay, we need to include particular actions to the stubborn set. In XTA time can not elapse at an urgent (committed) location or if invariant is stopping time.

Definition 10 (Time Enabling Action). *An action $a \in A$ is a* time enabling action *at zero time state $s = (\boldsymbol{l}, Z, \nu)$ if executing a may cause time to elapse. Formally $\mathsf{tea}^\sharp(a, s)$ iff $(\exists l \in \mathsf{Pre}(a).\ l \in L^u \cup L^c) \bigvee (\forall \mu \in Z, d \in \mathbb{R}_{\geq 0}.\ \mu + d \models I(l) \implies d = 0)$.*

Consider again Figure 1a and the zero time state $s = ((l_{1,3}, l_{2,3}, l_{3,3}), x_1 = x_2 = x_3 = 1500)$ and actions $a_i = \{(l_{i,3}, \tau, x_i \geq 1500, x_i = 0, l_{i,0})\}$. The actions are time enabling actions i.e. $\mathsf{tea}^\sharp(a_i, s)$ for $i \in \{1,2,3\}$. Note that as long as a time enabling action is enabled, time can not elapse. Thus executing independent actions can not cause time to progress.

Lemma 2. *Let $s \in S$, $a \in \mathsf{En}(s)$ with $\mathsf{tea}^\sharp(a, s)$ and $Delay^\sharp_s \stackrel{\mathsf{def}}{=} \{a\} \cup \{a' \in A^\sharp_s \mid a \rightsquigarrow^\sharp_s a'\}$. Then $\forall s' \in S, w \in (A^\sharp_s \setminus Delay^\sharp_s)^*.\ s \xrightarrow{w} s' \wedge \mathsf{zt}(s) \implies \mathsf{zt}(s')$.*

4.3 Preserving Goal States

In order to satisfy Condition $\mathcal{R}$ from Definition 2, which ensures that the reduction preserves goal states, we need to include actions whose execution is necessary to reach a goal state.

Definition 11 (Interesting Actions for Properties). *For formula φ and state s such that $s \not\models \varphi$. The set $\varphi_s^\sharp \subseteq A_s^\sharp$ is defined recursively based on the structure of φ as given by the following table:*

Formula φ	$\varphi_s^\sharp$
l	$\{a \in A_s^\sharp \mid l \in \mathsf{Post}(a)\}$
deadlock	*pick* $a \in \mathsf{En}(s)$ *then* $\{a\} \cup$ $\{a' \in A_s^\sharp \mid (\mathsf{Pre}(a) \cap \mathsf{Pre}(a') \neq \emptyset) \vee (\mathsf{Parallel}(a, a') \wedge a \leftrightsquigarrow_s^\sharp a'\}$
$x \bowtie c$	$\{a \in A_s^\sharp \mid \exists op \in \mathsf{Update}(a).\ x \bowtie c \leftrightsquigarrow_s^\sharp op\}$
φ_v *for* $v \in V$	$\{a \in A_s^\sharp \mid \exists op \in \mathsf{Op}(a).\ \varphi_v \leftrightsquigarrow_s^\sharp op\}$
$\varphi_1 \wedge \varphi_2$	$(\varphi_i)_s^\sharp$ *for some* $i \in \{1, 2\}$ *where* $s \not\models \varphi_i$

Lemma 3. *Given a state s, a formula φ, and the set $\varphi_s^\sharp$. Then $\forall s' \in S, w \in (A_s^\sharp \setminus \varphi_s^\sharp)^*$. $s \xrightarrow{w} s' \wedge \mathsf{zt}(s) \wedge s \not\models \varphi \implies s' \not\models \varphi$.*

5 Computing Stubborn Sets in Uppaal

We shall first provide a high level algorithm to compute a reachability preserving reduction for networks of timed automata and then discuss details related to the implementation of our technique in the model checker Uppaal.

5.1 Algorithm

Assume a given network of XTA and reachability formula φ. During the reachability analysis, we repeatedly use Algorithm 1 at every generated state s to compute a reduction $\mathsf{St}^\sharp$ that satisfies the conditions from Definition 2. At Line 1, we output $\mathsf{En}(s)$ should the state s be non-zero time state, thus satisfying Condition $\mathcal{Z}$. Line 3 includes all actions that are relevant for the preservation of the reachabily of states that can delay or belong to the goal states. Together with Lemma 2 and Lemma 3 this ensures that Condition $\mathcal{D}$ and Condition $\mathcal{R}$ are satisfied. Finally, the while loop starting at Line 5 ensures Condition $\mathcal{W}$. The while loop considers an action $a \in \mathsf{St}_s^\sharp$, if this action is not enabled then it will include all necessary actions which can enable it. This is done by adding actions which modify the location vector at Line 11, or by adding actions which modify the guards in a at Line 14. In the case where action a is enabled then the for loop at Line 16 includes all actions that are not independent with a.

Additionally, note that the set $A_s^\sharp$ is finite and in each iteration the size of $\mathsf{St}_s^\sharp$ can only increase because the only operation applied to $\mathsf{St}_s^\sharp$ is union. In the worst case we have $\mathsf{St}_s^\sharp = A_s^\sharp$ and hence the algorithm terminates.

Theorem 3 (Total Correctness). *Let $\mathcal{N}$ be a network of XTA and φ a formula. Algorithm 1 terminates and $\mathsf{St}^\sharp$ is a reachability preserving reduction where $\mathsf{St}^\sharp(s)$ is the output of Algorithm 1 for every state $s \in S$.*

Algorithm 1 Computing conditional stubborn sets

Input Network $\mathcal{A}_1, \ldots, \mathcal{A}_n$, state $s = (\boldsymbol{l}, Z, \nu)$, and formula φ.
Output Conditional stubborn set $\mathsf{St}^\sharp_s$

if $\neg\mathsf{zt}(s)$ **then return** $\mathsf{En}(s)$;
compute $A^\sharp_s$ and $\varphi^\sharp_s$;
if $\forall a \in \varphi^\sharp_s.\ \neg\mathsf{tea}^\sharp(a, s)$ **then** pick $a \in \mathsf{En}(s)$ with $\mathsf{tea}^\sharp(a, s)$; $\varphi^\sharp := \varphi^\sharp \cup \{a\}$;
$W := \varphi^\sharp_s$; $R := A^\sharp_s$; $\mathsf{St}^\sharp_s := W$
while $W \neq \emptyset$ and $\mathsf{En}(s) \cap \mathsf{St}^\sharp_s \neq \mathsf{En}(s)$ **do**
    Pick $a \in W$; $W := W \setminus \{a\}$; $\mathsf{St}^\sharp_s := \mathsf{St}^\sharp_s \cup \{a\}$; $R := R \setminus \{a\}$;
    if $a \notin \mathsf{En}(s)$ **then**
        for all $e \in a$ **do**
            if $src(e)$ is not in $\boldsymbol{l}$ **then**
                for all $a' \in R$ **do**
                    if $src(e) \in \mathsf{Post}(a')$ **then** $W := W \cup \{a'\}$;
            if exists $g \in \mathsf{Guard}(\{e\})$ such that $s \not\models g$ **then**
                for all $a' \in R$ **do**
                    if $\exists r \in \mathsf{Update}(a').\ g \leftrightsquigarrow^\sharp_s r$ **then** $W := W \cup \{a'\}$;
    if $a \in \mathsf{En}(s)$ **then**
        for all $a' \in R$ **do**
            if $(\mathsf{Pre}(a) \cap \mathsf{Pre}(a') \neq \emptyset) \vee (\mathsf{Parallel}(a, a') \wedge a \leftrightsquigarrow^\sharp_s a')$ **then**
                $W := W \cup \{a'\}$;
return $\mathsf{St}^\sharp_s$;

5.2 Implementation Details

Algorithm 1 is inserted as a state successor filter after the state successors are computed. This filter passes through only the states that are the result of stubborn actions. To improve the efficiency, the stubborn set is computed only when the origin state is urgent and has more than one successor, otherwise the filter just forwards all successors without any reduction. In the following we describe a number of optimizations that we included in our implementation.

Reachable Actions. In previous sections we have defined $A = 2^E$, as the set of actions. This set is unnecessary large and unpractical. The set of *reachable actions* from s can be semantically defined as $A_s \stackrel{\mathsf{def}}{=} \{a \in A \mid \exists s', s'' \in S, w \in A^*.\ s \xrightarrow{w} s' \xrightarrow{a} s''\}$. Our goal is to compute the smallest set $A^\sharp_s$ such that $A_s \subseteq A^\sharp_s \subseteq A$. Computing a small set has the advantage that potentially less dependencies are introduced, additionally it will reduce the computation time of stubborn sets. We implemented a static analysis in order to compute the set $A^\sharp_s$. Our analysis exploits the fact that time can not elapse at a state s, and thus actions that require a delay to become enabled need not be included in $A^\sharp_s$. For the performance sake, the approximation $A^\sharp_s$ is computed in two steps. The first step is prior to state exploration and is only executed once. In this step for each edge we compute the set of edges it can reach without doing a delay operation. The starting edge is assumed to be enabled and thus we start

with all possible clock assignments in conjunction with the source invariant. If clocks are compared against constants, we add the constraints. Otherwise, if integer variables appear on the guards, we relax (loose) all the information on the affected clock. The second step is executed at every urgent state and it is done by using precomputed data structures from the previous step that collect for every enabled edge the set of edges it can reach and then composes them into actions.

Broadcast Channels. Many Uppaal models use broadcast channels, however the set of possible broadcast synchronizations is exponentially large in terms of the number of potential receivers (in contrast to linear complexity of handshake synchronizations) and hence untenable for larger networks. Instead of computing all possible synchronizations, we compute one super-action for each broadcast sending edge, combining all potential receiving edges from other processes—this serves as a safe over-approximation. Such combined treatment avoids exponential blowup of broadcast actions at the cost of overly-conservative dependency checks, which considers a super-set of associated variables instead of precise sets involving a particular subset of receiving edges. In addition to broadcast synchronizations, the static analysis also supports arrays and C-like structures by expanding them into individual variables. Array indices, references and functions calls are over-approximated by using the ranges from variable types.

Precomputed Data Structures. To make our implementation fast, we precompute a number of data structures required by our technique. Examples include, edges leading to locations, some property base sets, reachable edges from locations. In particular, in order to compute the dependence between actions, the associated variable sets are also precomputed in advance for each action. These variable sets are then used to construct a dependency matrix over all reachable actions, thus making the action dependency check a constant-time lookup during verification.

6 Experiments

Table 1 shows the results of our POR implementation applied on a number of industrial case studies[1]. The experiments were run on a cluster with AMD EPYC 7551 processor with the timeout of 10 hours (and 15GB of RAM) for all models except for SecureRideSharing where the timeout was 48 hours (and 200GB of RAM). The model instances are suffixed with a number indicating the increasing amount of parallel components (sensors and the like).

FireAlarm is a simplified version of IndustFireAlarm [11] for the communication protocol of a wireless sensor network from German Company SeCa GmbH as described in Section 1. The AGless300 corresponds to a requirement from EN-54 standard that a sensor failure is reported in less than 300 seconds. A stricter property AGless100 is added to evaluate the performance when a property does

[1] Reproducibility package `https://github.com/DEIS-Tools/upor`

Model	Query	without POR		with POR		reduction ratio	
		states	time sec.	states	time sec.	states	time
FireAlarm4	AGnotdeadlock	27	<0.01	22	<0.01	1.23	–
FireAlarm20	AGnotdeadlock	1048635	148.41	270	0.01	3883.83	14841
FireAlarm100	AGnotdeadlock	–	OOM	5350	5.18	–	–
IndustFireAlarm13	AGless100*	931496	97.57	24296	2.81	38.34	34.72
IndustFireAlarm15	AGless100*	3684136	571.75	27672	3.84	133.14	148.89
IndustFireAlarm17	AGless100*	14694312	2884.18	31496	5.09	466.55	566.64
IndustFireAlarm19	AGless100*	58734632	15878.47	35768	7.20	1642.1	2205.34
IndustFireAlarm30	AGless100*	–	OOM	67272	27.92	–	–
IndustFireAlarm100	AGless100*	–	OOM	585272	2753.54	–	–
IndustFireAlarm13	AGless300	3731370	439.50	102570	12.73	36.38	34.52
IndustFireAlarm15	AGless300	14742718	2570.36	116862	17.69	126.15	145.30
IndustFireAlarm17	AGless300	58784210	12833.69	132946	23.15	442.17	554.37
IndustFireAlarm19	AGless300	–	OOM	150822	32.83	–	–
IndustFireAlarm30	AGless300	–	OOM	281172	128.08	–	–
IndustFireAlarm100	AGless300	–	OOM	2380752	12715.08	–	–
IndustFireAlarm13	AGnotdeadlock	3731320	388.63	63618	4.96	58.65	78.35
IndustFireAlarm15	AGnotdeadlock	14742668	2215.16	65654	5.68	224.55	389.99
IndustFireAlarm17	AGnotdeadlock	58784160	11202.80	67818	6.47	866.79	1731.50
IndustFireAlarm19	AGnotdeadlock	–	OOM	70110	8.00	–	–
IndustFireAlarm30	AGnotdeadlock	–	OOM	85004	17.85	–	–
IndustFireAlarm100	AGnotdeadlock	–	OOM	270504	530.46	–	–
SecureRideSharing6	AGlessMaxFail	200141	2.23	200141	5.60	1	0.40
SecureRideSharing7	AGlessMaxFail	7223770	95.60	7223770	252.61	1	0.38
SecureRideSharing8	AGlessMaxFail*	85622469	1467.49	85622469	3691.46	1	0.40
SecureRideSharing9	AGlessMaxFail*	1961298623	43548.8	1961298623	106223.46	1	0.41
SecureRideSharing6	AGnotdeadlock	200141	3.05	184973	6.3	1.08	0.48
SecureRideSharing7	AGnotdeadlock	7223770	122.29	2428033	93.21	2.98	1.31
SecureRideSharing8	AGnotdeadlock	97539581	2058.40	39387328	1845.46	2.48	1.12
SecureRideSharing9	AGnotdeadlock	–	OOM	944892374	55481.09	–	–
TTAC4	AGnotdeadlock	12213203	308.40	11414483	379.51	1.07	0.81
TTAC5	AGnotdeadlock	217259289	6724.25	204152089	8679.56	1.06	0.77
TTPA6	AGnotdeadlock	668421	27.30	668421	55.82	1	0.49
TTPA7	AGnotdeadlock	3329080	166.34	3329080	337.06	1	0.49
TTPA8	AGnotdeadlock	18073077	1096.79	18073077	2229.04	1	0.49
FB14	AGnotdeadlock	98310	138.22	98310	139.5	1	0.99
FB15	AGnotdeadlock	196614	698.54	196614	702.61	1	0.99
FB16	AGnotdeadlock	393222	2794.58	393222	2788.83	1	1

Table 1: Experimental results. Satisfiability results agree for all queries. Queries with * were not satisfied. The reduction is the ratio of performance without POR and with POR. OOM indicates out of memory.

not hold. Results show exponentially increasing savings in both number of states and computation time.

The SecureRideSharing models a fault-tolerant, duplicate-sensitive aggregation protocol for wireless sensor networks [12,3]. This case study did not show reductions until special treatment for broadcast synchronizations and variable increments was implemented. The AGnotdeadlock property shows substantial reductions, and one instance times out when POR is not used, however for the AGlessMaxFail query the state space is not reducible and the verification time is more than doubled due to variables reverenced in the query.

The TTAC models a Timed Triggered Architecture protocol [13] used in drive-by-wire vehicles. The TTPA models a Time-Triggered Protocol for SAE

class A sensor/actuator networks [26]. The model FB models the Field Buss scheduling protocol [24]. These case studies were selected as they do not allow for any state space reduction, thus allowing us to observe the time-overhead of our method. This overhead varies from almost no overhead for the FB models to twice as slow for the TTPA models.

7 Conclusion

We presented an application of partial order reduction based on stubborn sets to the model of network of timed automata in the Uppaal style, including a detailed analysis of both clock and discrete variable dependencies among the different components. The method allows us to reduce the state space in the situations where a sequence of mutually independent actions is performed while the network is in an urgent configuration where time cannot elapse (caused by the fact that at least one component is in urgent/committed location or there is a clock invariant imposing the urgency). Our method is implemented in the tool Uppaal and the experiments confirm that for the models with enough independent concurrent behavior in urgent situations, we can achieve exponential speedup in the reachability analysis. For models with limited urgent behavior, the overhead of our method is still acceptable (with the worst-case ratio of about 0.4 slowdown). These results are highly encouraging, yet further optimizations can be achieved by a more detailed static analysis of independent actions, one of the directions for future research.

Acknowledgments. We thank Christian Herrera and Sergio Feo Arenis for providing the models we use in our experimental section.

References

1. L. Aceto, A. Ingolfsdottir, K. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*, page 195. Cambridge University Press, 2007.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, Apr. 1994.
3. S. F. Arenis and B. Westphal. Formal verification of a parameterized data aggregation protocol. In G. Brat, N. Rungta, and A. Venet, editors, *NASA FM*, volume 7871 of *LNCS*, pages 428–434. Springer, 2013.
4. C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.
5. G. Behrmann, P. Bouyer, K. G. Larsen, and R. Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *STTT*, 8(3):204–215, 2006.
6. G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. In M. Bernardo and F. Corradini, editors, *SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer, 2004.
7. J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *CONCUR'98*, volume 1466 of *LNCS*, pages 485–500. Springer, 1998.
8. F. Boenneland, P. Jensen, K. Larsen, M. Muniz, and J. Srba. Start pruning when time gets urgent: Partial order reduction for timed systems. In *CAV'18*, volume 10981 of *LNCS*, pages 527–546. Springer-Verlag, 2018.

9. M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF toolset. In *Int. School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM-RT'04)*, volume 3185 of *LNCS*, pages 237–267. Springer, 2004.
10. D. Dams, R. Gerth, B. Knaack, and R. Kuiper. Partial-order reduction techniques for real-time model checking. *Formal Asp. Comput.*, 10(5-6):469–482, 1998.
11. S. Feo-Arenis, B. Westphal, D. Dietsch, M. Muñiz, and A. S. Andisha. *The Wireless Fire Alarm System: Ensuring Conformance to Industrial Standards through Formal Verification*, pages 658–672. Springer International Publishing, Cham, 2014.
12. S. Gobriel, D. Mosse, J. Brustoloni, and R. Melhem. Ridesharing: Fault tolerant aggregation in sensor networks using corrective actions. the 3rd annual. *IEEE SECON*, 2:595 – 604, 10 2006.
13. K. Godary. *Validation temporelle de réseaux embarqués critiques et fiables pour l'automobile*. PhD thesis, Ins. National des Sc. Appliquées de Lyon, France, 2004.
14. P. Godefroid, J. v. Leeuwen, J. Hartmanis, G. Goos, and P. Wolper. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, volume 1032 of *LNCS*. Springer-Verlag, 1996.
15. H. Hansen, S. Lin, Y. Liu, T. K. Nguyen, and J. Sun. Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions. In *CAV'14*, volume 8559 of *LNCS*, pages 391–406. Springer, 2014.
16. C. Herrera, B. Westphal, S. Feo-Arenis, M. Muñiz, and A. Podelski. Reducing quasi-equal clocks in networks of timed automata. In *FORMATS*, number 7595 in LNCS, pages 155–170. Springer, Heidelberg, 2012.
17. L. M. Kristensen, K. Schmidt, and A. Valmari. Question-guided stubborn set methods for state properties. *FM in System Design*, 29(3):215–251, 2006.
18. D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.
19. M. Minea. Partial order reduction for model checking of timed automata. In *CONCUR'99*, volume 1664 of *LNCS*, pages 431–446. Springer, 1999.
20. M. Muñiz, B. Westphal, and A. Podelski. Detecting quasi-equal clocks in timed automata. In *FORMATS*, pages 198–212, 2013.
21. P. Niebert and H. Qu. Adding invariants to event zone automata. In *FORMATS'06*, volume 4202 of *LNCS*, pages 290–305. Springer, 2006.
22. D. Peled. All from one, one for all: Nn model checking using representatives. In *(CAV'93)*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.
23. M. Perin and J. Faure. Coupling timed plant and controller models with urgent transitions without introducing deadlocks. In *17th International Conference on Emerging Technologies & Factory Automation (ETFA'12)*, pages 1–9. IEEE, 2012.
24. N. Petalidis. Verification of a fieldbus scheduling protocol using timed automata. *Computing and Informatics*, 28:655–672, 01 2009.
25. R. B. Salah, M. Bozga, and O. Maler. On interleaving in timed automata. In *CONCUR'06*, LNCS, pages 465–476, 2006.
26. W. Steiner and W. Elmenreich. Automatic recovery of the TTP/A sensor/actuator network. In *WISES*, 2003.
27. A. Valmari and H. Hansen. Stubborn set intuition explained. In *Transactions on Petri Nets and Other Models of Concurrency XII*, pages 140–165. Springer, 2017.